REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 02-05-2007	**2. REPORT TYPE** Final Scientific Technical Report	**3. DATES COVERED** *(From - To)* April 1, 2005 to December 31, 2006

4. TITLE AND SUBTITLE The DARPA Adaptive and Reflective Middleware Systems (ARMS) Program: Phase II Pervasive Instrumentation and Adaptation for Distributed Real-Time Embedded Systems: Final Technical Report	**5a. CONTRACT NUMBER** NBCHC030132 **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Balakrishnan Dasarathy	**5d. PROJECT NUMBER** 3GAR17 (Telcordia Internal Project Number) **5e. TASK NUMBER** 2 **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Telcordia Technologies, Inc. One Telcordia Drive Piscataway, NJ 08854-4157	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) The Defense Advanced Research Projects Agency, Information Exploitation Office, Attn: Dr. Joseph Cross, Program Manager, Room 635, 3701 North Fairfax Dr., Arlington, VA 22203-1714	**10. SPONSOR/MONITOR'S ACRONYM(S)** DARPA IXO **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This is the final technical report on the Adaptive and Reflective Middleware Systems (ARMS) Phase II work by a team led by Telcordia. Our focus in the ARMS program is in the development of an adaptive and reflective network Quality of Service (QoS) infrastructure for the Total Ship Computing Environment (TSCE) of next generation surface ships. Our technology uses a Bandwidth Broker to provide admission control, and leverages Differentiated Services and Class of Service functionality of high-end routers and switches for enforcement. In the Phase II ARMS program, we built upon our Phase I network QoS technology to provide continued assurance of network QoS for mission-critical tasks in the presence of certain catastrophic faults such as losing an entire data center, and improve the timely adaptation to network performance with probes and instrumentation for delay.

15. SUBJECT TERMS
adaptive reflective middleware, network QoS, admission control, bandwidth brokering, mission-critical systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		15	Balakrishnan Dasarathy
					19b. TELEPHONE NUMBER *(Include area code)* 732-699-2430

Telcordia.

The DARPA Adaptive and Reflective Middleware Systems (ARMS) Program: Phase II

Pervasive Instrumentation and Adaptation for Distributed Real-Time Embedded Systems: Final Technical Report

Telcordia Technologies

Contract No.: NBCHC030132

February 5, 2007

Abstract: This is the final technical report on the Adaptive and Reflective Middleware Systems (ARMS) Phase II work by a team led by Telcordia. Our focus in the ARMS program is in the development of an adaptive and reflective network Quality of Service (QoS) infrastructure for the Total Ship Computing Environment (TSCE) of next generation surface ships. Our technology uses a Bandwidth Broker to provide admission control, and leverages Differentiated Services and Class of Service functionality of high-end routers and switches for enforcement. In the Phase II ARMS program, we built upon our Phase 1 network QoS technology to provide continued assurance of network QoS for mission-critical tasks in the presence of single-mode and certain catastrophic faults such as losing an entire data center and improve the timely adaptation to network performance with probes and instrumentation for delay. Our work also raises the level of abstraction for configuration and deployment of distributed real-time embedded systems, specifically for achieving end-to-end QoS in such systems, using Model-Driven Development (MDD) tools.

1. Introduction

This is the final technical report on the Adaptive and Reflective Middleware Systems (ARMS) Phase II work by the Telcordia team. The Telcordia team consists of Telcordia Technologies as prime contractor and Vanderbilt University and Prism Technologies (PrismTech) as subcontractors. The work was performed from April 1, 2005 to December 31, 2006. The report discusses architecture, implementation and validation aspects of the technology developed during ARMS Phase II. The report also includes a discussion of deliverables and key results.

This Phase II work built on Telcordia Phase I work on Quality of Service (QoS) assurance for layer-3/layer-2 networks using the Multi-Layer Resource Management (MLRM) architecture framework [1]. The MLRM framework is a major output from Phase I of the ARMS program and resulted from the combined efforts of the program participants [1]. Its goal is to demonstrate that adaptive and reflective middleware can make substantial improvement in the effectiveness of the Total Ship Computing Environment (TSCE) used in the DDG 1000 program.

In Phase I, we developed and demonstrated the basic building blocks of an adaptive and reflective network QoS technology. Our adaptive QoS technology uses a Bandwidth Broker to provide admission control and enforcement using the Differentiated Services (DiffServ) and Class of Service (CoS) functionality of high-end COTS routers and switches. The Bandwidth Broker technology detects and adapts to changes in mission requirements, work load, and configuration. It uses discovery algorithms to maintain a current view of resource availability and traffic probes to detect the changing needs of high-priority flows. In the Phase II ARMS program, we built upon this network QoS technology to

- provide continued assurance of network QoS for mission-critical tasks in the presence of single-mode and certain catastrophic faults such as losing an entire data center or a pool of resources;
- improve the timely adaptation to network performance with probes and instrumentation for delay, jitter, and available bandwidth; and
- increase the flexibility of our delay guarantees by incorporating deadline support in flow admission decisions based on sound mathematical calculations.

Moreover, our proposed network QoS solution also addressed policy changes in resource management in response to operational mode changes (e.g., normal to alert mode), typical of battlesphere environments, specifically those affecting network QoS globally. We also raised the level of abstraction for configuration, deployment, and testing of distributed real-time embedded systems, specifically for achieving end-to-end QoS in such systems, using Model-Driven Development (MDD) tools.

The effectiveness of our technology, specifically for recovery from network faults, operating in conjunction with the technologies of other program participants was validated through an ARMS Phase II gate metric, known as Gate 3.

We next describe in detail our technical approach.

2. Technical Approach

2.1. Network QoS Components

Figure 1 illustrates our overall network architecture. In Figure 1 R/S, ASM, IA and PM stand for Router/Switch, Application String Manager, Infrastructure Allocator and Pool Manager, respectively. ASM, IA and PM are higher-level MLRM components that are users of the network quality of service functionality provided by the Bandwidth Broker. The major logical components of the network QoS management architecture are:

- Bandwidth Broker
- Flow Provisioner
- Performance Monitor
- Fault Monitor

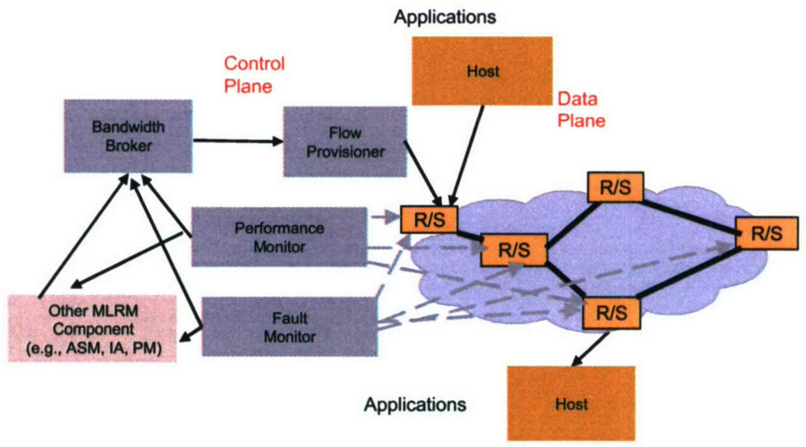

Figure 1: Network QoS Architecture

We now describe these four components:

Bandwidth Broker: Higher level MLRM components use the basic functions provided by the Bandwidth Broker to allocate and schedule mission tasks spanning the network. These functions include:

- Flow Admission Functions: Reserve, commit, modify, and delete flows (in support of distributed scheduling); and
- Queries: Retrieve bandwidth availability for different classes between pairs of pools and subnets (in support of allocation of processes to processors).

See [1], [3] for more details on the Bandwidth Broker.

The Bandwidth Broker solution leverages DiffServ in layer-3 and CoS mechanisms in layer-2 network elements, in a transparent manner, to provide end-to-end QoS guarantees in a hybrid, heterogeneous environment. CoS mechanisms provide functionality at layer 2 similar to what DiffServ mechanisms provide at layer 3. They both provide aggregated traffic treatment in the core of the network and per-flow treatment at the edge of the network. Typical network implementations of QoS using DiffServ/CoS perform the following steps:

1. At the ingress of the network, traffic is classified and marked as belonging to a particular class and may be policed or shaped to ensure that it does not exceed a certain rate or deviate from a certain profile.
2. In the network core, traffic is placed into different classes based on the marking of individual packets. Each class is provided treatment differentiated from all other classes but consistent for all packets within the class. This includes scheduling mechanisms that assign weights or priorities to different traffic classes (such as weighted fair queuing or priority queuing, respectively), and buffer management techniques that include assigning relative buffer sizes for different classes and packet-discard algorithms such as Random Early Detection (RED) and Weighed Random Early Detection (WRED).

DiffServ/CoS features by themselves are insufficient to guarantee end-to-end network QoS, because the traffic presented to the network must be made to match the network capacity. The main function of the Bandwidth Broker then is adaptive and reflective admission control that ensures there are adequate network resources to match the needs of admitted flows. To do its job, the admission control entity needs to be aware of the path being traversed by each flow, track how much bandwidth is being committed on each link for each traffic class, and estimate whether the traffic demands of new flows can be accommodated. As such, path discovery in combined layer-2 and layer-3 network was a major area of focus in Phase I.

In our approach, the Bandwidth Broker is also responsible for overall coordination. For instance, the Bandwidth Broker is responsible for assigning the appropriate traffic class to each flow, and coordinating the provisioning of complex parameters for policing, marking, scheduling, and buffer management, such that contracted flows obtain the promised end-to-end QoS.

<u>Support for Delay Bounds</u>: In Phase I, the Bandwidth Broker admission decision for a flow was based on whether or not there was enough bandwidth on each link traversed by the flow. Toward the end of Phase I, we developed the computational techniques to provide both deterministic and statistical delay bound guarantees. These guarantees are based on relatively expensive computations of occupancy or utilization bounds for various classes of traffic, performed only at the time of network configuration/reconfiguration, and relatively inexpensive checking for a violation of these bounds at the time of admission of a new flow. (See [4] for details.) As alluded to earlier, delay guarantees raise the level of abstraction of the Bandwidth Broker as seen by the higher layer MLRM components and enable these components to provide better end-to-end mission guarantees. In Phase II, we implemented these computational techniques to provide both deterministic and statistical delay bound guarantees as part of the Bandwidth Broker admission control process.

In Phase II, we also developed computational techniques for increasing the abstraction of the Bandwidth Broker in an area, known as bulk scheduling [5]. This involves supporting admission requests that specify an aggregate number of bytes ("bulk") to be delivered and a deadline for the delivery of the full aggregate. This research explored some of the issues that arise in trying to get a network to ensure that the transfer of a given bulk be completed within a given time. Mainly, the task is to identify the policing-parameter values necessary to support a TCP session, but the appropriate values are affected by certain details of the implementation, such as the nature of the assurance being sought, and whether traffic shaping is being done in addition to policing. In the case of shaping, the required buffer space also needs to be determined. This work identified the quantitative implications of these details in a variety of cases, including the implications for the occupancy at which the network can be run.

<u>Non-Blocking Admission Control</u>: There was one major concern expressed regarding the Bandwidth Broker during our technology transition effort. For certain classes of high priority traffic or certain users with low delay tolerance, the delay associated with

admission control may be intolerable. Our solution, non-blocking admission control, alleviates this problem by allowing high priority traffic to begin using network resources while the admission control process runs its course. This approach allows us to manage resources in the network during periods of scarcity while minimizing the impact on applications during periods when no scarcity exists. In the transient period between the flow initiator's admission control request and the admission control response from the Bandwidth Broker, the network may be over-subscribed. Our solution encompasses several strategies that minimize the impact to the admitted flows in the network during this transient period. These strategies include pre-allocating bandwidth for use during transient periods and using best effort for transient periods for (non-blocking) requests.

Flow Provisioner: The Flow Provisioner translates technology-independent configuration directives generated by the Bandwidth Broker into vendor-specific router and switch commands to classify, mark, and police packets belonging to a flow. In Phase I, we implemented Flow Provisioners for layer-3 IOS CISCO (e.g., CISCO 3600 routers), layer-2/3 Catalyst switches (e.g., CISCO 6500 switches) and layer-2/3 IOS switches (e.g., CISCO 4507 switches) to demonstrate the viability of this QoS architecture in a variety of network topologies and equipment. In Phase II, our work focused on needed extensions to support a metric known as Gate 1 or the "Do No Harm" metric. This metric dealt with how well MLRM technologies supported reverting back to static modes. The Flow Provisioner was extended to support provisioning operations that revert the network QoS configuration settings to their original state (i.e., their state prior to any Bandwidth Broker admission control decisions).

Performance Monitor: In Phase I, our monitoring was limited to detection of overflow of traffic for an admitted flow. This feature is useful in detecting mission-critical tasks that require additional bandwidth. However, much needs to be done here for our infrastructure to be continuously adaptive and reflective. We need features to monitor how well critical flows are meeting their timing constraints, specifically their end-to-end-latency and jitter metrics. Salient features of the Performance Monitor we have implemented are:

- Delay measurements can be collected for specific traffic flows or for all flows between a pair of hosts belonging to a specific traffic class. Averaging window sizes can be specified. The Performance Monitor interface supports both synchronous requests to query current delay and asynchronous events for violations of thresholds or to provide periodic updates on delay.
- The analysis and management of performance data is separated from probes for raw data collection.
- Probe job configurations are stored in a persistent medium so as to recover the job data in case of probe platform or probe host failures.
- In setting up a probe job, we have options to vary packet size, gap/time between packets, the number of packets in a packet train and periodicity.
- The probe job packet train generation is done at the (Linux) kernel level to control or minimize the time-related vagaries in generating packets.

- Clocks are synchronized between two measurement (Linux) hosts with GPS using a non-network interface between the hosts to achieve delay measurement accuracy in the micro-second range.

Fault Monitor: A key feature of a resource management system in a dynamic battlespace environment is the ability to detect and react to network faults. We illustrate the problem we are trying to address using the network shown in Figure 2.

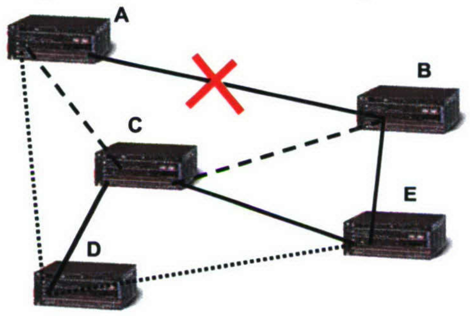

Figure 2: Impact of a Fault on QoS

If the link between switches A and B goes down, then a flow Y between A and B may be routed through switch C (shown in dashed lines). Similarly, a flow Z between E and A that originally used the links EB and BA may now use links ED and DA (shown in dotted lines). However, links AC, CB, ED, and DA may now be oversubscribed causing concerns on QoS guarantees for Y and Z as well as for the flows that had been using these links prior to the occurrence of the fault.

The goal of the Fault Monitor is not to perform a root cause analysis or enable fixing the fault, but to do QoS restoration. If the QoS of a previously admitted flow cannot be guaranteed, the Fault Monitor will raise a fault exception event to the Bandwidth Broker. The Bandwidth Broker, in turn, will raise a higher-level event to other MLRM components, such as ASM and PM. The three functional aspects of the Fault Monitor components are described next.

- Fault Detection: SNMP traps are used to detect link failures (and links coming back into service). A switch failure is detected when SNMP trap notifications for all links to the switch are received by the adjacent switches.
- Impact Analysis: For each admitted flow the impact analysis involves determining whether the flow has changed its path using the path discovery algorithms. If a path for a flow has changed, that flow has been impacted by the failure and is a candidate for re-admission.
- QoS Restoration: Our design is capable of supporting different algorithms satisfying different utility functions or optimality criteria. The first step, regardless of the algorithm used, is to temporarily remove affected flows. In the current implementation, the affected flows are then readmitted one at a time, from the highest priority to the lowest priority and within the same priority with less bandwidth first. Our goal is to readmit the maximum number of higher priority flows whose paths have changed. We may substitute an algorithm that admits more flows. We can also employ a preemption algorithm. For instance, if the admission of a flow would lead to capacity violations on a link, then the process

preempts a lower priority flow that uses this link. The preempted flow then becomes a candidate for readmission.

The Fault Monitor also tracks the paths and the resources used when the fault disappears and restores guarantees to the original level. We carry out the entire process of impact analysis and restoration, as described above, when the fault condition disappears. See [8] for more details on the Fault Monitor component.

Software Fault Tolerance: A technology area of focus is recovery of the various network QoS components from a single process or processor failure. The recovery of the Bandwidth Broker and Flow Provisioner is of particular concern, as they are responsible for changing the state of the network (by provisioning various network elements for desired QoS behavior) and they maintain a view of the network. The Fault Monitor and Performance Monitor are feedback mechanisms, and, as such, recovery of these components is less of a concern.

The Bandwidth Broker is a "database" application. The relational DBMS MySQL is the persistence mechanism used to recover from process and processor failures. The database is used to maintain the current commitment of network resources on a continual basis. The persistent information includes network inventory, currently admitted flows, paths taken by flows between various end-points of the network, configuration details such as access list numbers used for each flow, and lastly DSCP value allotment for each flow. Apart from this network data, for each host that uses Bandwidth Broker services, the Bandwidth Broker also maintains a record that indicates where the host is connected to the network and to which gateway IP address its traffic is directed by default. Within the Bandwidth Broker code there are two transactional scopes. When a request to add or delete a flow is processed, the first transactional scope essentially consists of database updates to track the committed bandwidth correctly on each link traversed by the flow. At the end of this commit process, the record corresponding to the flow request is updated with a "provisioning in progress" status indicator. The second transaction block surrounds a call to the Flow Provisioner to provision the network element. At the end of this transaction scope the status of the flow record is changed from "provisioning in progress" to "completed."

The Bandwidth Broker, Flow Provisioner and the MySQL database are all replicated to improve recovery time. However, a request is served by only one instance of these components at any time. This is known as the warm-passive approach to replication. With the warm-passive approach, there is no replica start up time involved, and, moreover, the computations involved can be non-deterministic, i.e., the technique is more widely applicable than the active replication approach. If the Bandwidth Broker fails during the first transactional scope, the CORBA middleware will raise an exception to the client and the client will retry the entire request using the second instance. To gracefully recover from the failure during the second transactional scope, the second instance of the Bandwidth Broker will start honoring the request by re-executing the instructions in the second transactional scope. We employ a modified group Communication and CORBA fault tolerance middleware mechanism that provides the needed transparency in binding an MLRM client to the right instance of the Bandwidth Broker during recovery. We have implemented replication of the database using the MySQL Cluster technology. (See http://dev.mysql.com/doc/refman/4.1/en/ndbcluster.html). The MySQL Cluster technology supports in-memory databases and the various replicas in a cluster are updated synchronously, i.e., the replicated instances are always consistent at the completion of a transaction.

The Flow Provisioner is designed as a stateless component. A failure in an instance of the Flow Provisioner is handled by the Bandwidth Broker reissuing commands for all "in progress" requests to routers/switches (using the secondary instance of the Flow Provisioner). However, instructions to routers are not always idempotent. When an instruction to a router is not idempotent, a "delete/undo" operation needs to precede a non-idempotent provisioning instruction.

2.2. Support for Mission Mode Changes

In addition to adapting to faults and overload, the Bandwidth Broker supports dynamically changing modes. A mode is a major operational situation such as normal, alert, or battle mode in a military environment. Our work in support of mode changes deals with global policy changes affecting the entire network, including changes in the fraction of the bandwidth allocated to various traffic classes. The bandwidth policy change implementation involves sending reconfiguration instructions to every switch to change its QoS parameters such as queue size, number of scheduling slots allocated, and packet drop rules for every traffic class. Such a policy change often results in the reduction of the bandwidth allocated to one or more traffic classes so that QoS for various flows already admitted in these classes might no longer be guaranteed. Identifying and readmitting the affected flows is similar in spirit to the impact analysis and restoration of QoS in response to network faults, but the details are somewhat different. A flow is affected in this case if there is a link in its path for which the total bandwidth allocated to the link exceeds the link capacity of the flow's class and/or the current occupancy value of the flow's class exceeds its corresponding threshold for the class. For the flows affected, the primary sorting field is priority, from the lowest priority to highest priority, and within each priority we sort on the bandwidth size in descending order. We delete the flows in the affected list starting from the one with the lowest priority and highest bandwidth requirement until there is no link in the path used by the flow such that the total bandwidth allocated for the link (for that class) exceeds the link capacity (for that class) and/or the current occupancy value exceeds its corresponding threshold. The utility function used here minimizes the number of higher priority flows that have the potential of being denied their QoS. When a flow is deleted, the bandwidth used or the current occupancy value in all the links used by the flow needs to be adjusted accordingly.

2.3. Expanding the Scope of Applicability

To prepare our QoS technology for transition to several DoD programs, we expanded its scope of applicability. Two areas were of focus: Support for IPv6 and Operation in a Multi-Level Security environment.

Support for IPv6: Our goals here were to:

- Make our network QoS management tools run in and for an IPv6 environment
- Leverage the Flow Label field of IPv6 for enhanced network QoS differentiation and control
- Provide tools and techniques to transition IPv4-based middleware/network applications to run in an IPv6 environment

To make our software run in an IPv6 environment, our investigation has concluded that changes to our software are required to:

- Reflect the fact that OSPF Link-State Advertisements have changed in IPv6;
- Deal with multiple addresses per interface; and
- Address IP address length and subnet to host relationships in our database tables.

The IPv6 Flow Label field helps with tracking of individual end-to-end flows using a 20-bit field. IPv4 flow classifiers are based on the 5-tuple of the source and destination addresses, ports, and the transport protocol type. However, some of these fields may be unavailable due to either fragmentation or encryption, and locating them past a chain of IPv6 option headers may be inefficient. The usage of the Flow Label along with the source and destination address fields enables efficient IPv6 flow classification, where only the IPv6 main header fields in fixed positions are used. Many more flows can be distinguished and tracked using (source address, destination address, Flow Label ID) than using (source address, destination address, DSCP). Moreover, the requests to the Bandwidth Broker can be simplified by allocating Flow Label IDs to applications. For instance, video applications may have a different set of Flow Label ID's from audio applications. So, in the request to the Bandwidth Broker, certain QoS parameters (type of service, rate, burst size) for the sessions need not be specified with the use of Flow Label IDs.

In the area of tools and techniques for transitioning applications and middleware to IPv6, the toolsets required can be classified into three classes:

- Tools for Discovery of IP-centric features of code: Allows understanding of the patterns of IP usage. It is particularly important to discover whether IP data is used at the application/middleware level (e.g. ftp)
- Tools for remediation of code: Enables isolating the areas where such patterns occur and replacing the IPv4-specific code with IPv6-capable code where needed
- Tools for testing of remediated code: Ensures that the previous two steps, discovery of IP-centric feature related code and their remediation, worked correctly.

Bandwidth Architecture for Multi-Level Security: The purpose of this work is to 'bake security into' our Bandwidth Broker based network QoS. The architecture we have developed minimizes the need for trusted interfaces as much as possible. Several scenarios of bandwidth reservation admission requests and bandwidth reservation cancel requests have been worked out to ensure correct communication in each case. See [6].

2.4. Integration Research Issues

The network QoS solution we have proposed is not a stand-alone solution; rather it is part of an end-to-end MLRM solution. As such, we needed to integrate and test our solutions with those of others in the ARMS programs using the QoS-enabled component middleware. A research agenda here was to achieve a measure of automation in the deployment, configuration, testing and integration of application components using model-driven development (MDD) technology and using the MLRM technology. We worked very closely with Vanderbilt University who led the research effort. Although the QoS-enabled component middleware they previously developed offers many desirable features, until recently it has lacked the ability to simplify and coordinate application-level services to leverage advances in end-to-end network QoS management.

Working with Vanderbilt University, we developed a declarative framework called NetQoPE that integrates modeling and provisioning of network QoS with QoS-enabled component middleware for enterprise DRE systems. NetQoPE enhances their prior work that predominantly considered only host resources when provisioning end-system QoS. The enhanced system integrates with the Bandwidth Broker to provide network QoS in the QoS-enabled middleware. NetQoPE's modeling capabilities now allow users to (1) specify application QoS requirements in terms of network resource utilization needs, (2) generate deployment plans that account for the network resources, and (3) provision network and host elements to enable and enforce the network QoS decisions. See [7] for details.

2.5. Bandwidth Broker and Real-Time

Our network QoS components are Java/CORBA server applications; they depend on a variety of third party middleware (JacORB, MySQL, log4j, OpenCCM). We conducted some preliminary experiments with the Bandwidth Broker with an eye toward making it 'more real time' without significant code changes and toward applying the lessons learned to other Java server applications that need to run in a mission critical environment [9]. Our experiments demonstrate that there is there no significant variance in the service time for requests to the Bandwidth Broker. The occasional spikes in the elapsed time could be due to the operating system (OS), in which case a RT OS could reduce them. They could also be due to a service (MySQL) used by the Bandwidth Broker. Unfortunately, we were not able to instrument in such a way that we could determine the culprit. It does not appear that a real-time garbage collector or real-time ORB would have helped in our case. Tools to simplify the analysis process, however, would help considerably. Overall, the experiment was successful in showing that the Bandwidth Broker already has fairly good soft real time characteristics which could be improved by some redesign. In addition, our analysis techniques show promise for other, similar Java applications.

3. *Deliverables*

We have successfully met the various deliverable requirements throughout the performance period. Our main software/prototype deliverables this period are the Performance Monitor and the Fault Monitor. Several enhancements were also made to the Bandwidth Broker and Flow Provisioner. The highlights of our software deliverables are:

- Network Performance Monitor to measure latency with delay accuracy in the micro-second range
- Network Fault Monitor to detect and restore network QoS in the case of single faults and certain catastrophic faults, such as outage of a data center in a ship
- Fault tolerant implementation of Bandwidth Broker and Flow Provisioner using MySQL Cluster database technology (The MySQL cluster technology needs to be enhanced to support recovery needs that are suitable for hostile military environments that were considered in ARMS.)
- Enhancements to the Bandwidth Broker to support mode changes affecting resource management policies throughout the network
- Enhancements to the Flow Provisioner to support "Do No Harm" metric
- Enhancement to the Bandwidth Broker to support delay bound guarantees for mission-critical traffic
- Enhancements to the Bandwidth Broker to support non-blocking admission control for short-duration mission-critical traffic
- Integration of the network resource management into QoS-enabled middleware so as to raise the level of abstraction and automation toward achieving end-to-end QoS for CORBA Component Model (CCM)-based DRE systems

In addition, we have developed (and documented) a multi-level security architecture for our network components that minimizes the amount of communications through guards. We have also developed computational techniques in support of scheduling bulk traffic that has a strict latency constraint through the network.

We produced several technical reports/papers during Phase II:

- A Middleware-based Network QoS Provisioning Engine for Enterprise Distributed Real-time and Embedded Systems [7]
- Fast Recovery and QoS Assurance in the Presence of Network Faults [8]
- QoS in the Presence of Multi-Level Security Network Architecture (Presentation), [6]
- Adaptive Network QoS in Layer-3/Layer-2 Networks as a Middleware Service for Mission-Critical Applications [1]
- Meeting Deadlines for Bulk Transfers [3]
- Toward a 'More Real Time' Bandwidth Broker [9]

4. Experimentation and Validation

The goal of our main area of experimentation was to demonstrate that our network QoS software can recover well under the limit set by the DDG 1000. There are two gate tests under Gate 3, known as Gate 3A and 3B.

- Gate Test 3A was defined to exhibit that we could provide fault tolerance for the ARMS MLRM that meets the DDG 1000 requirements for fault tolerance of their resource management system. Specifically, the requirement states that the recovery from a single pool/data center failure should be within one second.

Gate Test 3B was defined to exhibit that we could make the ARMS MLRM recover from faults beyond those required by DDG 1000, specifically that we could survive the failure of two MLRM instances (the operational one and a partially recovered replacement) in rapid succession. The metrics for Gate Test 3B ask (1) can MLRM recover from two cascading pool failures and (2) within what time?

In the context of our network QoS components, only the Bandwidth Broker and Flow Provisioner were of focus in this study as they are responsible for changing the state of the network by provisioning various network elements for desired QoS behavior and as they maintain a view of the network, as mentioned previously. The recovery time of interest in the case of Gate 3A is the time taken by a client of the Bandwidth Broker (e.g., ASM) to switch from the primary instance of the Bandwidth Broker (and Flow Provisioner) to the secondary instance of the Bandwidth Broker and get a response back from the secondary instance when the failure occurs during the processing of the transaction by the primary Bandwidth Broker. The primary and secondary Bandwidth Broker instances (and their corresponding Flow Provisioner instances) were in different datacenters or pools of resources. The result for five Gate Test 3A runs on Emulab (www.emulab.net) is as given in Table 1.

Average recovery time (ms)	212.48
Minimum recovery time (ms)	150.10
Maximum recovery time (ms)	283.20
Standard Deviation (ms)	52.39

Table 1: Gate 3A Recovery Time

In the Gate 3B experiments, we injected the first fault to cause a primary data center failure in the same manner as Gate Test 3A, and then injected the second fault (so as to cause a failure in a secondary that is trying establish itself as the primary) in as close to the worst case time as we could, i.e., after the system is far along in its recovery, but just before it is done recovering so that the faults cannot be handled as two distinct failures.

In all the Gate 3B experiments, an MLRM client using the Bandwidth Broker was able to recover from both failures. The result for five Gate Test 3B runs on Emulab, as given in Table 2 below, indicates that the sub-second recovery from two cascading failures is achievable.

Average recovery time (ms)	509.06
Minimum recovery time (ms)	414.90
Maximum recovery time (ms)	579.60
Standard Deviation (ms)	50.85

Table 2: Gate 3B Recovery Time

In Phase I, we conducted experiments to demonstrate that the Bandwidth Broker and Flow Provisioner can perform effective dynamic resource management to increase mission capacity in Emulab. We repeated these experiments in a larger scale at the Vanderbilt ISIS Lab to evaluate (1) the effectiveness of various traffic classes supported by the Bandwidth Broker, specifically High Priority, High Reliability, Multi-Media, and Best Effort, (2) admission control mechanisms that ensure there is enough capacity for the flows that have been admitted with QoS guarantees, and (3) underlying network element mechanisms that police a flow for compliance, i.e., that prevent a flow from exceeding its allocated bandwidth amount. The results are summarized and discussed in [7].

5. Results

This Phase II work built upon Telcordia Phase I work on QoS assurance for layer-3/layer-2 networks using the Multi-Layer Resource Management (MLRM) architecture framework. The MLRM framework was a major output from Phase I of the ARMS program and resulted from the combined efforts of the program participants. The purpose of the MLRM architecture is to push middleware technologies beyond current commercial capabilities. The current capabilities are largely limited to fixed static allocation of resources in support of predefined mission capabilities. Static allocations limit the ability of a military application to adapt to conditions that vary from the original system design. It is desirable for resource allocation to be performed dynamically and modified in response to faults, to changes in mission requirements, or to workload distributions that do not match the original mission-planning model.

In Phase I, we developed and demonstrated the basic building blocks of an adaptive and reflective network QoS technology. In Phase II, we built upon this Phase I work. The Bandwidth Broker technology now adapts to changes in mission requirements, work load, and resource availability.

Moreover, our network QoS solution now addresses mode change policies, specifically those affecting network QoS globally. A mode here is taken to mean a major operational situation such as normal, alert, or battle mode. We also proposed a security enforcement architecture that would minimize the amount of "talking down" communications through guards among our network QoS components in a multi-level security environment. The benefits of the proposed work compared to the current state-of-the-art and alternative approaches include:

- The work has led to adaptive and reflective network resource management integrated into an overall adaptive and reflective resource management system for the Total Ship Computing Environment which will enable more effective use of a ship's computing resources in dynamically changing and possibly hostile circumstances. Compared to the pre-existing static approach, this offers the potential for more effective execution of the ship's mission. A major issue remaining for effective technology transition is certification of such dynamic systems.
- Our delay bounds calculations are set in a more generalized framework than what is found in the literature. Our calculations support any number of priority classes and within a priority class any number of weighted fair queuing classes. We support both deterministic and statistical guarantees using this generalized framework.

Deterministic guarantees are applicable to the highest priority tasks, and statistical guarantees are more broadly applicable to all lower priority tasks. Our implementation for delay support in the Bandwidth Broker in ARMS Phase II is flexible and unique in the mix of guarantees it can provide -- deterministic delay guarantees, statistical delay guarantees, and capacity (bandwidth-based admission control) guarantees -- to various tasks.

- Our solution provides an integrated QoS treatment for heterogeneous layer-2 and layer-3 networks, can be centrally directed and policy-driven, and is more scalable than another commonly used QoS technique. The two main technologies for providing differentiated treatment of traffic are DiffServ/CoS and Integated Services (IntServ). The Bandwidth Broker makes use of DiffServ/CoS. In IntServ, every router makes the decision whether or not to admit a flow with a given QoS requirement. Some drawbacks with conventional implementations of IntServ are that (1) it requires per-flow state at each router, which can limit its scalability; (2) it makes its admission decisions based on local information rather than some adaptive, network-wide policy; and (3) it is applicable only to layer-3 IP networks.

6. References

[1] B. Dasarathy, S. Gadgil, R. Vaidyanathan, K. Parmeswaran, B. Coan, M. Conarty, and V. Bhanot, Network QoS Assurance in a Multi-Layer Adaptive Resource Management Scheme for Mission-Critical Applications using the CORBA Middleware Framework, *Proc. RTAS 2005 (11th IEEE Real-Time and Embedded Technology and Application Symposium)*, pp. 246-255, March 7-10, 2005, San Francisco, CA.

[2] Lardieri, P., J. Balasubramanian, D.C. Schmidt, G. Thaker, A. Gokhale, and T. Damiano, 2006. A Multi-layered Resource Management Framework for Dynamic Resource, Management in Enterprise DRE Systems, Elsevier Journal of Systems and Software, Accepted for Publication at a Special Issue on Dynamic Resource Management in Distributed Real-Time Systems, Edited by Charles Cavanaugh, Frank Drews and Lonnie Welch.

[3] Balakrishnan Dasarathy, Shrirang Gadgil, Ravi Vaidyanathan, Arnie Neidhardt, Brian Coan, Kirthika Parmeswaran, Allen McIntosh, and Frederick Porter, Adaptive Network QoS in Layer-3/Layer-2 Networks as a Middleware Service for Mission-Critical Applications, Accepted for Publication at a Special Issue on Dynamic Resource Management in Distributed Real-Time Systems, Edited by Charles Cavanaugh, Frank Drews and Lonnie Welch.

[4] Arnie Neidhardt, Brian Coan, and Balakrishnan Dasarathy, Calculations for Admission Control, Telcordia ARMS Phase I Report, January 12, 2005.

[5] Arnie Neidhardt, Meeting Deadlines for Bulk Transfers, April 2006.

[6] Scott Alexander and B. Dasarathy, QoS in the Presence of Multi-Level Security Network Architecture, August 2006 (Presentation).

[7] Jaiganesh Balasubramanian, Sumant Tambe, Shrirang Gadgil, Frederick Porter, Aniruddha Gokhale, Douglas C. Schmidt, Balakrishnan Dasarathy and Nanbor Wang, NetQoPE: A Middleware-based Network QoS Provisioning Engine for Enterprise

Distributed Real-time and Embedded Systems, November 2006 (Submitted to ICDCS 2007).

[8] Shrirang Gadgil, Balakrishnan Dasarathy, Frederick Porter, Kirthika Parmeswaran, and Ravi Vaidyanathan, Fast Recovery and QoS Assurance in the Presence of Network Faults, Telcordia ARMS Phase II Report, October, 2006.

[9] Frederick Porter, Toward a 'More Real Time' Bandwidth Broker, May 2006.